View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Universiteit Twente Repository

Run-time Mapping of Applications to a Heterogeneous SoC

Lodewijk T. Smit, Johann L. Hurink and Gerard J. M. Smit
University of Twente, Enschede, The Netherlands

***Abstract*— This paper presents an iterative hierarchical approach to map an application to a parallel heterogeneous SoC architecture at run-time. The application is modeled as a set of communicating processes. The optimization objective is to minimize the energy consumption of the SoC, while still providing the required Quality of Service. This approach is flexible, scalable and the performance looks promising.**

I. Introduction

Due to short time to market and reuse of designs, SoC architectures that are composed of commercially of the shelf available intellectual property (IP) blocks are becoming popular. Ultimately, we would like to have a SoC architecture that is flexible enough to run different applications (within a certain application domain). However, mapping an application to such a heterogeneous SoC is more difficult compared to a homogeneous architecture.

Common practice is to map the applications to the architecture at design-time. In this paper we consider how to perform the mapping at run-time. Run-time mapping offers a number of advantages over design-time mapping. It offers the possibility:

- to adapt to the available resources. The available resources may vary over time due to applications running simultaneously or adaptation of algorithms to the environment.
- to enable unforeseeable upgrades after first product release time, e.g. new applications and new or changing standards.
- to avoid defective parts of a SoC. Larger chips mean lower yield. The yield can be improved when the mapper is able to avoid faulty parts of the chip. Also aging can lead to faulty parts that are unforeseeable at design-time.

II. Problem Description

This article describes a mapper that maps applications to a heterogeneous SoC architecture at run-time. A number of inputs are required for the mapper: a description of the applications (Section II-A), a library of process implementations (Section II-B) and a description of the architecture (Section II-C).

A. Application Description

An application is assumed to be given by a set of processes and interconnections between processes needed for communication. We believe that in practice this partitioning is done manually by an experienced designer, and do no consider this process in this paper.

Additional application Quality of Service requirements for the application such as required timing behaviour (e.g. throughput, latency) are also specified in the application description.

B. Library Description

For each process of an application, one or more process implementations have to be provided. A process implementation is the implementation of a process on a particular tile, e.g. object code for an ARM or a DSP or configuration data for an FPGA.

A process implementation has several characteristics, e.g. the amount of energy it takes to execute the process on a particular tile of the architecture (see section below). Other examples are delay or tile utilization. This library (including the characteristics) is composed at design-time. In our approach, only the selection of process implementations is done at run-time.

C. Architecture Description

The heterogeneous SoC architecture consists of multiple tiles of different types (e.g. ARM, FPGA, DSP) interconnected by a Network-on-Chip (NoC). For each tile a number of characteristics has to be provided on beforehand, such as the type of the tile, the amount of available memory, the clock frequency, etc.

The NoC consists of routers and links. The links are used to interconnect routers or a tile with a router. It is possible to have different links in parallel between the same source and destination. Also the NoC characteristics have to be provided, such as the topology of the network, the frequency of the clock of the network, latency per router, etc.

D. Goal

The objective of the mapper is to determine a mapping of the application(s) to the architecture using the library at run-time. The mapping has to minimize the energy consumption and has to satisfy all the constraints of the application and the architecture. E.g. the application may need real-time guarantees; constraints of the architecture are for example limited capacity of processing tiles and links.

0-7803-9294-9/05/$20.00 ©2005 IEEE

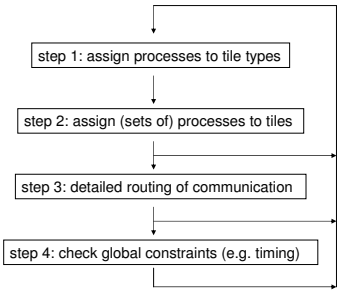

Fig. 1. Hierarchical Iterative Approach

III. Approach

When we only consider the subproblem of mapping processes to available tiles (without considering the consequences for the communication), this can be formulated as a General Assignment Problem (GAP). The GAP is NP-complete [2] and, thus, for realistic problem sizes the computational effort to solve the problem to optimality at run-time gets too large. Therefore, heuristics have to be used to find a solution with a reasonable quality within an acceptable time.

To deal with the complexity of the problem, we propose a hierarchical iterative approach. The idea is to solve the problem using multiple levels. At each level a particular decision is made that shrinks the search space. Decisions of previous levels are considered to be fixed at the lower levels. On higher levels not all details are taken into account to improve the speed of evaluation. In other words, higher levels use a higher abstraction.

This hierarchical approach has the danger, that decisions which seems to be promising on a higher level due to the underlying assumptions of the abstraction level, show to be bad or even lead to resulting subproblems which are infeasible. However, this becomes clear only on the lower levels. Therefore, we propose to evaluate the initial solution (= mapping achieved after considering all levels once) and to backtrack to higher levels and make suggestions to improve the solution on that level.

A. Levels

This section describes the four levels or steps that are distinguished in our iterative hierarchical model to solve our mapping problem. Figure 1 shows the used levels.

1 The goal of the first step is to assign processes to a tiletype and for each tiletype a partitioning of its processes in as many subsets as there are tiles of the type available. This partitioning takes into account that each set does not exceed the capacity of its tiletype. Tiles of the same type are assumed to have the same capacity. A set may contain multiple processes if the tile allows time multiplexing. An empty set implies an idle tile. Note that in this level the sets are not assigned to a specific tile, but only to a tile type.
The method to achieve this goal is as follows. Start with empty sets (the number of sets is equal to the number of tiles). Iteratively assign a process to a set. The choice of the next process to be assigned depends on the difference in costs of assigning the process to the different tile types. More precisely, the difference between the cheapest assignment and the second cheapest assignment reflects the desirability of assigning the process now. In other words, if the alternative is more expensive the desirability to map the process now increases. If a process has been chosen to be assigned next, this process is not only assigned to the tiletype, but also to one of the corresponding sets. This is done in a greedy manner by assigning it to the first set where it fits due to its capacity constraints.
Consider a SoC with 5 DSPs. For example, it is decided in step 1 that process number 1 is mapped to a DSP, but not which particular DSP. The process is added to one of the 5 sets that contain the assigned processes for a DSP in such a way that the load associated with each set stays below the capacity of a DSP.

2 We have as result from step 1: sets belonging to tile types. In the second step we want an assignment of these sets to concrete tiles. During the assignment the communication costs are take into account, but not the capacity restrictions within the NoC.
The method to achieve this is split up in two procedures. In the first procedure, we do not change the sets resulting from step 1, but swap sets that are assigned to each tile type. This is done in a local search manner by considering swaps of sets and executing the swap which gives the most gain.
In the second procedure, we change the assignment of an individual process between sets. Hereby, we do not change the given assignment of sets to tiles, which resulted from procedure 1. Also, this is solved by a local search procedure. All possible reinsertions of processes to different sets (also for other tiletypes that are allowed for the processes) are evaluated. For the evaluation of a reinsertion the change in communication as well as computation costs are considered.
These two procedures may be used in different manners. One may iterate these two procedures several times using the resulting assignments as starting solution for the next step.
Another possibility is not to execute the result of the second procedure in this step, but to backtrack to step 1. In this backtracking, reinsertion of a process to a set of a different tiletype will change the assignment costs in step 1. If it appears that it is more beneficial to run a process on another tile, feedback is provided to the first step. This feedback assigns bonus points for mapping this process to this tile. The advantage of the second approach is that not only small local changes are executed but that on base of the new information another global assignment of processes to tiletypes is generated.
Using the second approach a process can not be moved to a tile of a different type during step 2, which is done in step 1 only. In step 1 the communications costs are neglected. However, it may appear to be more efficient to choose a tile with higher computational costs if the reduction in communication costs outperforms the

increase in computation costs for this tile. This step can 'repair' this by providing feedback to the higher levels. This information is used by the higher levels in a next iteration.

3 For the concrete realization of step 3 the channels are sorted by non increasing throughput. Next, iteratively for each channel a corresponding path is determined taking into account the loads resulting from the previously mapped channels. The sorting is done to increase the probability that a heavy demanding channel gets assigned a better path. In each iteration for a given channel a shortest path between the source and destination tile of the channel has to be determined, where only such routers and links are taken into account which still have enough capacity for the throughput of the current channel. Currently, the communication costs are reflected by the throughput multiplied by the lengths of the path. If several shortest paths exist, the path which leaves the largest remaining capacity on it is chosen. This makes it easier to check the timing guarantees in step 4.
   During the calculation of the shortest path for some channel it may happen that it is detected that no path exists. In this case it is possible to look for the bottlenecks and give some feedback to a higher level and generate a solution on that level taking into account this feedback (e.g. routing may fail due to resource shortage in the NoC. Remapping one process may solve this problem).

4 The last step checks all global constraints. The most important global constraint is timing. Note that these are global requirements that surpass the requirements of the individual components such as processes and channels. For example, if the communication takes longer due to a longer path in the network, the consequence may be that the processing has to be done faster to satisfy the timing requirements.
   If bottlenecks are identified in this step, feedback may be generated to a higher level. For example, it may be suggested to run a process on a processor with less delay.

In general, the production of feedback immediately triggers a new iteration to prevent that multiple changes influences the mapping process.

It is important to realize that this proposed iterative hierarchical approach differs significantly from simple local search methods that are often used in heuristics. The feedback from a lower level may result in a complete different mapping on a higher level in a next iteration.

IV. Application Example

This section describes a realistic application that is partitioned into communicating processes.

Digitale Radio Mondiale (DRM) [1] is a standard for digital radio below the 30 Mhz. DRM is a OFDM based system using a multi-level convolutional coding scheme for error correction and MPEG-4 audio coding for source coding. A concise explanation of the DRM standard can be found in [3]. This section describes the mapping of a part (baseband processing) of a DRM receiver to a heterogeneous tiled SoC architecture

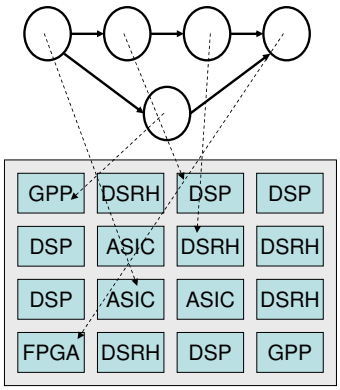

Fig. 2. SoC Template and the Mapping of a Process Graph

with 16 processing tiles. We use the SoC template that is depicted in the bottom of Figure 2. The top of Figure 2 shows a set of communicating processes. The arcs suggests a possible mapping.

We number the tiles in the SoC as follows: The left top tile is tile number 0, the next right neighbor is tile number 1, the right bottom tile is tile number 15.

Figure 3 shows the processes of the digital baseband part of our DRM receiver. Table I shows the processes that we would like to map on the SoC (for a functional description of the processes see [3]). These processes concern the data flow of the DRM application; we do not consider the processes 9,10,11 in the "Global control & estimation" part of Figure 3. To test our algorithms, it is not crucial to have very accurate estimations of the execution costs. Therefore, to save implementation time, the concrete values for the processing and communication costs used in our example are based on estimations and not the result of concrete measurements.

- the number of multiplications per second is used as an indication for the costs of a process. Table I shows the costs of the process in terms of multiplications per second for reception of Mode B, and the available implementations for the different type of processors.
- the ratio between processing a multiplication on an ASIC, DSRH, FPGA, DSP, GPP are assumed to be 10:40:50:60:500 respectively.
- the communication costs increase linearly with the distance of the communication path on the SoC. The communication costs are equal to the throughput in kbit/s given in Table II multiplied by the Manhattan distance between the tiles.

Note that processes that have an ASIC realization need a specific ASIC. It is not possible to assign a process with an ASIC realization to an arbitrary ASIC processor. For real systems, a more accurate estimation of the costs is required.

V. Mapping Results

We implemented a mapper with step 1 - 3 of the proposed hierarchical iterative approach in C. We ran our mapping algorithm for the described DRM example. Execution of a functional not optimized version (including all initializations) on an ARM processor took about 4 million instructions. This is

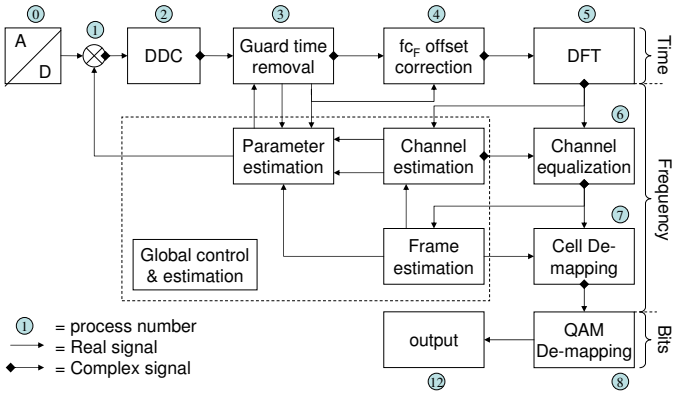

Fig. 3. DRM Processes to Map on SoC

equivalent to a 2-3 milliseconds on a 100 Mhz ARM processor. The results of the mapper are compared with:

- the optimal solution that is computed by exhaustive enumeration of all possible mappings using quadratic programming [6]. This computation took 10 hours on a Pentium 4 PC.
- an algorithm called 'Adapted MinWeight' [4], [5]. This heuristic is based on a dynamic programming approach but the dynamic programming principle is slightly modified to make it possible to add the capacity constraint for the tiles.

Table III shows the results (the assignments and the total costs) of the different algorithms. In the mapping, index i (starting at index zero) denotes the mapping of the ith process to a certain processornumber. So, e.g. for all mappings, process 3 (the fourth process) is assigned to processor 12.

The hierarchical iterative approach comes up with the optimal solution as shown in Table III. This was achieved with only three iterations that used feedback that was produced

TABLE I
MULTIPLICATION COSTS FOR DRM MODE B

Block	Process	Multiplies	Processors
A/D converter	0	0	ASIC
Mixer	1	24k	DSP, DSRH, GPP
DDC	2	0	ASIC
Guard time correlation	3	144k	DSP, FPGA, GPP
Frequency Correction	4	96k	DSP, DSRH, GPP
FFT	5	346k	ASIC, DSP, DSRH, GPP
Channel equalization	6	38k	GPP, DSP, DSRH
Demapping	7	0	GPP, DSP, DSRH
Bit decoding	8	0	GPP, DSP, DSRH
Output	12	0	GPP

TABLE II
COMMUNICATION COSTS FOR DRM MODE B

Edge	kbit/s
$0 \rightarrow 1$	375k
$1 \rightarrow 2$	750k
$2 \rightarrow 3$	755k
$3 \rightarrow 4$	600k
$4 \rightarrow 5$	600k
$5 \rightarrow 6$	300k
$6 \rightarrow 7$	241k
$7 \rightarrow 8$	201k
$8 \rightarrow 12$	47k

TABLE III
DIFFERENT MAPPINGS

algorithm	mapping	costs
Adapted MinWeight	5, 13, 9, 12, 6, 10, 7, 3, 2, 0	24126
Quadratic programming	5, 1, 9, 12, 13, 10, 6, 7, 11, 15	22954
Hierarchical approach	5, 1, 9, 12, 13, 10, 6, 7, 11, 15	22954

in the preceeding iteration. In general, we may not expect that our approach will always find the optimal solution, but the performance shown for this particular example is quite encouraging to continue research in this direction.

The MinWeight algorithm provides a mapping that is about 5% worse compared to the optimum mapping. This is still good. However, the main drawback of the MinWeight algorithm is the lack of flexibility and scalability of the approach. This means that we can not add additional constraints easily and the algorithm will have a longer execution time for larger problems because the state space grows fast. This is the reason to change from the dynamic programming approach to the described iterative hierarchical approach.

VI. CONCLUSION AND FUTURE WORK

The hierarchical approach with relative simple heuristics in each individual level looks promising to solve the mapping problem fast at run-time.

The following open issues are subject to future work:

- Design and implementation of step 4 (check global constraints).
- Take (smart) action if an infeasible solution is created in step 1 or 3. Currently, the mapper stops with a message that no mapping can be found.
- Detect and prevent oscillations due to feedback.
- Generate more application examples for testing.

Acknowledgement

This research is conducted within the FP6 Smart ChipS for Smart Surroundings (4S) project (IST-001908) supported by the European Commission and within the Adaptive Ad-hoc Free Band Wireless Communications (AAF) project supported by FreeBand Communications.

REFERENCES

[1] E. T. S. I. (ETSI). Digital radio mondial (drm); system specification, Apr. 2003. ETSI ES 201 980 v1.2.2 (2003-04).

[2] M. L. Fisher, R. Jaikumar, and L. N. V. Wassenhove. A multiplier adjustment method for the generalized assignment problem. *Management Science*, 32(9):1095–1103, 1986.

[3] F. Hofmann, C. Hansen, and W. Schäfer. Digital radio mondiale (drm) digital sound broadcasting in the AM bands. *IEEE Transactions on Broadcasting*, 49(3):319–328, Sept. 2003.

[4] L. T. Smit, G. J. M. Smit, J. L. Hurink, H. Broersma, D. Paulusma, and P. T. Wolkotte. Run-time assignment of tasks to multiple heterogeneous processors. In *Proceedings of the 4rd PROGRESS workshop on embedded systems*, pages 185–192, Oct. 2004.

[5] L. T. Smit, G. J. M. Smit, J. L. Hurink, H. Broersma, D. Paulusma, and P. T. Wolkotte. Run-time mapping of applications to a heterogeneous reconfigurable tiled system on chip architecture. In O. Diesel and J. A. Williams, editors, *Proceedings of the International Conference on Field-Programmable Technology*, pages 421–424, Dec. 2004. ISBN: 0-7803-8651-5.

[6] W. L. Winston. *Operations Research; Applications and Algorithms*. International Thomson Publishing, 3 edition, 1993. ISBN:0-534-20971-8.